Uso de XGBoost para problemas de clasificación

Laura G Funderburk

Sobre mí

🤓 Completé la Licenciatura en Matemáticas en Simon Fraser University, Canadá
📈🤗 Científica de datos, Developer Advocate @Ploomber
🇲🇽 🇬🇧 🇩🇪 🇨🇦 Nací en México, he vivido en Inglaterra, Alemania y ahora en Canadá
🥋 Practico Brazilian JiuJitsu

Conecta conmigo

🖥️ Sitio web: **https://lfunderburk.github.io/**
🐙 GitHub: **https://github.com/lfunderburk**
🐦 Twitter: @lgfunderburk
🦣 Mastodon: **https://fosstodon.org/@lfunderburk**

Introducción

1. ¿Qué es la clasificación?
2. Diferencia entre clasificación y regresión
3. Introducción a XGBoost

Qué es la clasificación

Clasificación vs Regresión

1. Clasificación: predecir una etiqueta categórica (ej. si es spam o no es spam)
2. Regresión: predecir un valor numérico continuo (ej. el precio de bienes raices)

XGBoost

- Basado en algoritmo de Gradient Boosting
- Escalable y eficiente
- Ampliamente utilizado en competencias de aprendizaje automático

Instalación de XGBoost

```
pip install xgboost
```

Importando bibliotecas

In [1]:
```
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, \
                            classification_report, \
                            accuracy_score,\
                            balanced_accuracy_score,\
                            ConfusionMatrixDisplay

import pandas as pd
```

Cargando y dividiendo el conjunto de datos

- **Conjunto de datos "Calidad del Vino"**: propiedades fisicoquímicas de vinos, como acidez, contenido de azúcar y alcohol.
- **Objetivo**: predecir la calidad del vino usando estas características.
- **Problema de clasificación binaria (buena o mala calidad)**.

Este conjunto de datos está disponible en el Repositorio de Aprendizaje Automático de UCI.
https://archive.ics.uci.edu/ml/datasets/wine+quality

In [2]:
```
# Carga el conjunto de datos
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winec
df = pd.read_csv(url, sep=';')

# Preprocesamiento: convierta la columna de calidad en un problema de clasificación binari
threshold = 6
df['quality'] = (df['quality'] >= threshold).astype(int)

# Dividir el conjunto de datos en características (X) y objetivo (y)
X = df.drop('quality', axis=1)
y = df['quality']

X.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.

In [3]:
```
y.head()
```

Out[3]:

```
0    0
1    0
2    0
3    1
4    0
Name: quality, dtype: int64
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                random_state=42)
```

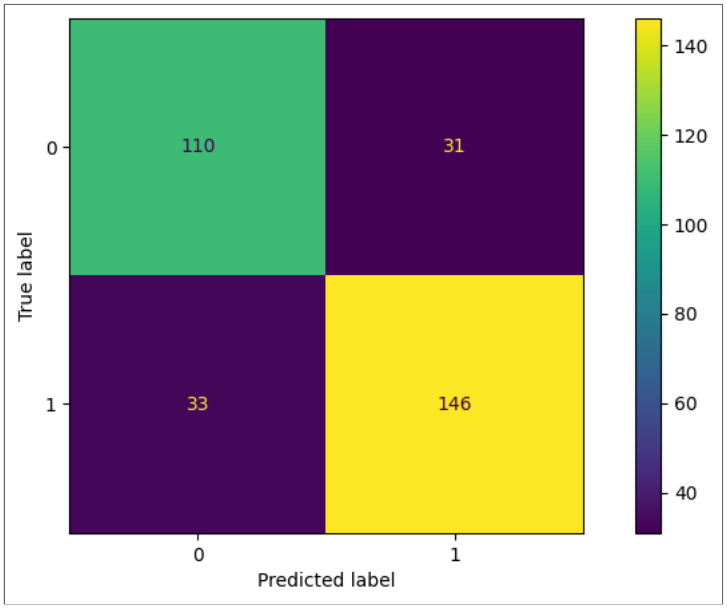
Entrenamiento del modelo

In [5]:
```
# Inicializa el modelo
modelo_xgb = xgb.XGBClassifier()

# Ajustar el modelo
modelo_xgb.fit(X_train, y_train)

# Usa el modelo para predecir
y_pred = modelo_xgb.predict(X_test)
```

Evaluación del modelo

In [6]:
```
def evalua_modelo(modelo):
        score_train = modelo.score(X_train, y_train)
    score_test = modelo.score(X_test, y_test)
    balanced_accuracy = balanced_accuracy_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    print('score for training set', score_train, 'score for testing set', score_test)
    print("Balanced accuracy score", balanced_accuracy,"Accuracy",accuracy)
    fig, ax = plt.subplots(figsize=(10, 5))
    ConfusionMatrixDisplay.from_predictions(y_test, y_pred, ax=ax);

evalua_modelo(modelo_xgb)
```

```
score for training set 1.0 score for testing set 0.8
Balanced accuracy score 0.797892151036095 Accuracy 0.8
```



Hiperparámetros

- `learning_rate`
- `max_depth`
- `n_estimators`
- `subsample`
- `colsample_bytree`

In [7]:
```
# Definiendo hiperparámetros personalizados
hiperparametros = {
    'learning_rate': 0.1,
    'max_depth': 3,
    'n_estimators': 100,
    'subsample': 0.8,
    'colsample_bytree': 0.8
}

# Entrenamiento del modelo con hiperparámetros personalizados
modelo_xgb_h = xgb.XGBClassifier(**hiperparametros)
modelo_xgb_h.fit(X_train, y_train)
```

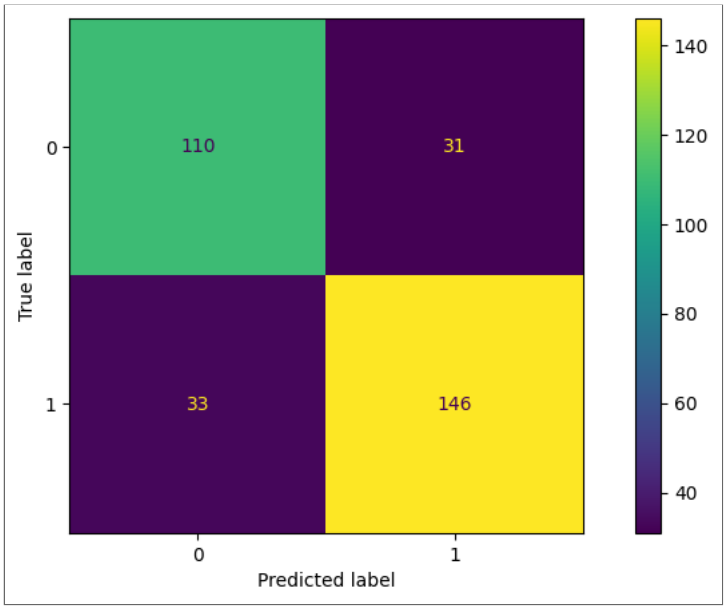
Out[7]:

▼ XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [8]:
```
# Observar resultados
evalua_modelo(modelo_xgb_h)
```

```
score for training set 0.8569194683346364 score for testing set 0.771875
Balanced accuracy score 0.797892151036095 Accuracy 0.8
```



Ajuste de hiperparámetros

- Búsqueda de cuadrícula
- Búsqueda aleatoria
- Validación cruzada

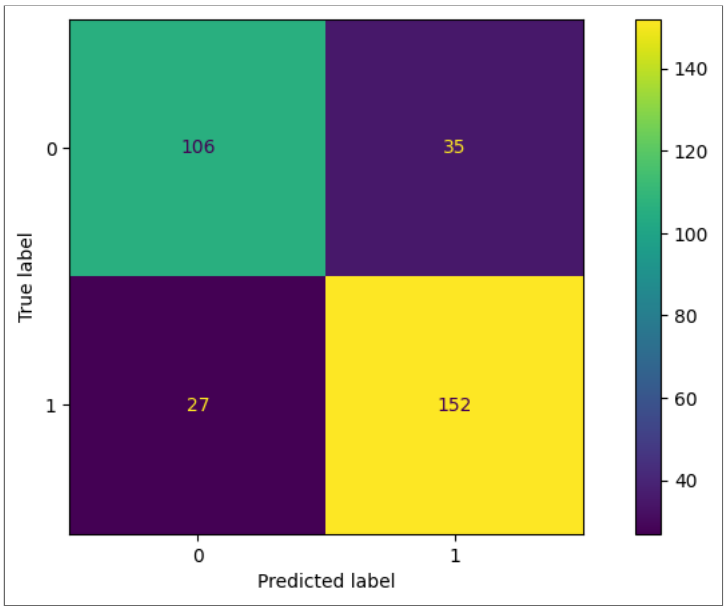
In [9]:
```
# Hyperparameters ranges
    hyperparameters_ranges = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': list(range(3, 15)),
    'n_estimators': [50, 100, 200, 500, 1000],
    'subsample': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
}
```

In [10]:
```
from sklearn.model_selection import RandomizedSearchCV

# Create XGBoost classifier
modelo_xgb = xgb.XGBClassifier()

# Perform randomized search
random_search = RandomizedSearchCV(modelo_xgb,
                                   param_distributions=hyperparameters_ranges,
                                   n_iter=25,
                                   cv=5,
                                   random_state=42)
random_search.fit(X_train, y_train)

# Get the best hyperparameters
best_hyperparameters = random_search.best_params_
print(f"Best hyperparameters: {best_hyperparameters}")
```

```
Best hyperparameters: {'subsample': 0.9, 'n_estimators': 1000, 'max_dept
h': 8, 'learning_rate': 0.01, 'colsample_bytree': 1.0}
```

In [11]:
```
# Train the model with the best hyperparameters
modelo_xgb_h_b = xgb.XGBClassifier(**best_hyperparameters)
modelo_xgb_h_b.fit(X_train, y_train)

# Evaluate the model
y_pred = modelo_xgb_h_b.predict(X_test)

evalua_modelo(modelo_xgb_h_b)
```

```
score for training set 1.0 score for testing set 0.80625
Balanced accuracy score 0.8004675304092872 Accuracy 0.80625
```

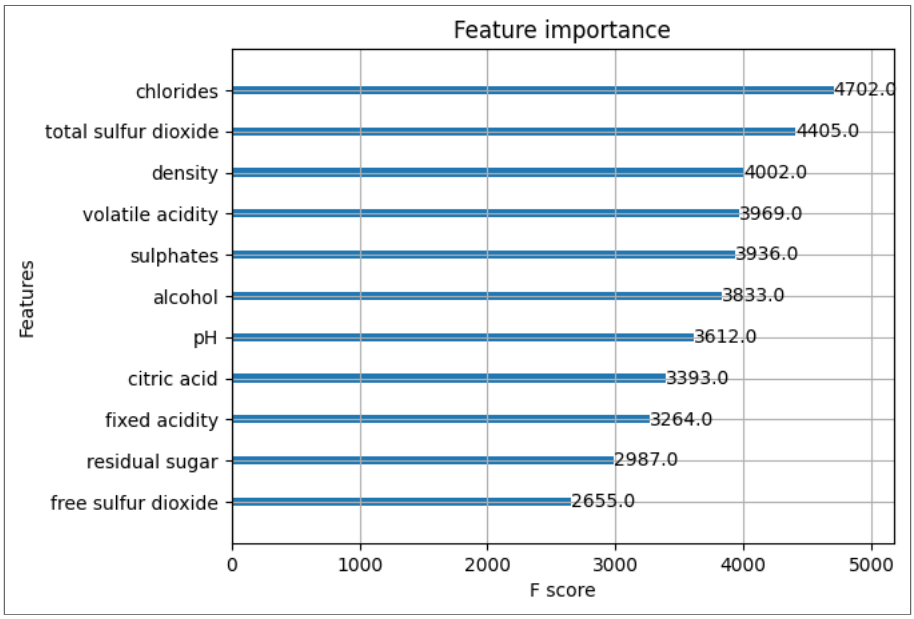


Importancia de las características

- Ganancia
- Frecuencia
- Cobertura

Visualización de la importancia de las características

```
In [12]: xgb.plot_importance(modelo_xgb_h_b)
         plt.show()
```



Ventajas de XGBoost

- Rendimiento superior
- Manejo de valores faltantes
- Paralelización
- Regularización
- Flexibilidad

Conclusión

- XGBoost: eficiente y potente para clasificación
- Ajuste de hiperparámetros y selección de características
- Ventajas clave de XGBoost

Conecta conmigo

🖥️ Sitio web: **https://lfunderburk.github.io/**
🐙 GitHub: **https://github.com/lfunderburk**
🐦 Twitter: @lgfunderburk
🦣 Mastodon: **https://fosstodon.org/@lfunderburk**